DYNAMICS OF RAILWAY NETS

Dines Bjørner

Computer Science and Engineering
Informatics and Mathematical Modelling
Building 322, Richard Petersens Plads
Technical University of Denmark
DK–2800 Kgs.Lyngby, Denmark
E–Mail: db@imm.dtu.dk, URL: www.imm.dtu.dk/˜db

Abstract: From an example formal (ie., computing science based) model of railway nets, their rail units and the various forms of states of these rail units, we put forward some suggestions for future research & development that engages researchers of automatic control and computing science in collaborative work. The background for our paper is the increasing practice of software engineers engaging ever more in what used to be considered traditional control engineering tasks — without, we claim, there being a clear, scientific basis for doing so.

1. THE PROBLEM

The problem to be tackled by this paper is one of "trying to come to grips" with, or, at least, identifying some issues of computing science that "lies close", it is believed, to issues of control theory, ie., theory of automation.

1.1 A Dichotomy

Computing Science, the study and knowledge of how to construct the "things" that can exist inside computers, is a young science. Also when compared to *Control Theory.* Control theory, for us "outsiders", seems to have more–or–less unanimously accepted the central rôle of mathematics in modeling actual world phenomena. From these mathematical domain models, control theory then goes on to identify the problem, ie., derive the requirements, and from this again, "derive" a solution to the problem — all of this strongly based on the use of non–trivial mathematics. Computing science has yet to register that kind of agreement on the indispensability of using mathematics — perhaps not so much, anymore, amongst researchers in academia, as among so–called practicing software engineers.

And software engineers clearly, in almost all cases, fail to build on proper, commonly agreed up mathematical domain models of railways. Computing science is placed in a tenacious "battle" — unknown, I dare say, to most computer and computing scientists, let alone to software engineers — between computer science[1] and software engineering. Todays software engineer rarely uses profound results, available since decades, in daily software development. Something unheard of in other engineering disciplines.

1.2 An Experiment — And Why ?

In this paper we will, not daunted by this sad state of affairs, nevertheless, take a look at what might be border–lines, overlapping, or nicely interfaced (time will show), between computing science and

[1] The study and knowledge of the properties of the "things" that can exist inside computers.

control theory. To do so we perform an experiment. One that is entirely within the realm of todays seemingly more advanced software engineering cum computing science. The experiment aims at constructing and presentng a model, as "we normally do it", of an application domain. That domain is one in which both software engineers and control engineers have a long, respectively very long interest. The domain is that of railway systems. The objective of the experiment is to be able to discuss some possible border–lines and interfaces between control theory and computing science.

1.3 The Experiment

We limit ourselves, prudently, to just considering railway nets. Basically as they are laid down, there, on the ground, through tunnels, on bridges, etc. And we limit ourselves to consider not so much the topologies of such nets — that was done in Bjørner (2000)[2] — but to consider some state properties of the "smallest" components of railway nets: The rail units, those "little" things that — when you buy a "toy" railway for yourself or your children, then — you buy: Straight and curved linear rail units, simple two–way switches and simple cross–overs !

We will look at how we model their behaviour over time. And we will relate such models to more general models such as pursued by control engineers.

1.4 Caveats

But we need to warn the reader, for fear of "overselling" what we are presenting: We model railway nets, first "as they are", later "as we would wish them to be". That is: First with all their possible states, whether desirable or not, then with those states that lead to safe traffic. From the point of view taken in that modelling we do not consider the details of the dynamics of the actual changing of a switch: We model that it takes time to switch. Say t_σ. But we do not model the actual dynamic behaviour of the various mechanical, electro–mechanical and electronic devices involved in that switching. So we do not really model their control ! Eventually we model that they be controlled; that they be switcched between specific states — and then leave it to control engineers to design the switch such that the torque and momentum of the various electro–mechanical gadgets be appropriate to secure a safe switch.

[2] The essence of the model of Bjørner (2000) is included here as appendix A.

1.5 Structure of Paper

The paper is organised as follows: In Section 2 we present, in several small steps, the model that we wish to relate to some aspects of control theory. The latter is done in Section 3. The many small steps of Section 2 proceed as follows: First, in Section 2.1 we cover the "statics" of rail units. In Section 2.2 we cover the main rail unit states that we decide to model: Section 2.2.1 models the types, ie., the classes of values that model time durations — for the various states. Section 2.2.2 models rail unit stable states — those during which trains may be routed through these units. Section 2.2.3 models the states that rail units are in when making simple transitions from stable states to stable states. And Section 2.2.4 models the states that are sometimes, perhaps, for most rail units, very rarely if ever attained — but in which reconfigurations of rail units may take place.

In Section 2.3 we model the time–span limited behaviour of units, seen in isolation; that is: Not as part of the net. Section 2.3.1 thus "lifts" from units to timed units. Section 2.3.2 models operations on timed units. while Section 2.3.3 reviews possible state sequences of timed units.

Finally Section 2.4 "lifts" nets to timed nets: In Section 2.4.1 we relate timed nets to "their" timed units. In Section 2.4.4 we briefly review properties that timed nets are expected to satisfy when subject to operations on timed units.

Finally, in Section 3, we discuss, on the background of the "large" example of Section 2, some possible relations between software and control engineering.

The formalisation is expressed in the `RAISE` [3] George et al. (1995) Specification Language: `RSL` George et al. (1992)[4] We explain this notation, through "footnoted" Annotations. Not by explaining the formal language constructs. By by explaining what specific formulas express. In that way the meaning of the formal language constructs is, so–to–speak, "smuggled" in.

2. THE RAILWAY NET EXAMPLE

The example thus illustrates the dynamics of rail nets. Figure 1 on the following page shows an 'abstracted' rail net. In our example we shall disregard the distinctions of stations, lines, tracks, sidings, and lines 'connecting' stations. We shall just concentrate on the "smallest" parts: The rail units.

[3] `RAISE` stands for Rigorous Approach to Idustrial Software Engineering
[4] URL: `http://www.iist.unu.edu./raise`

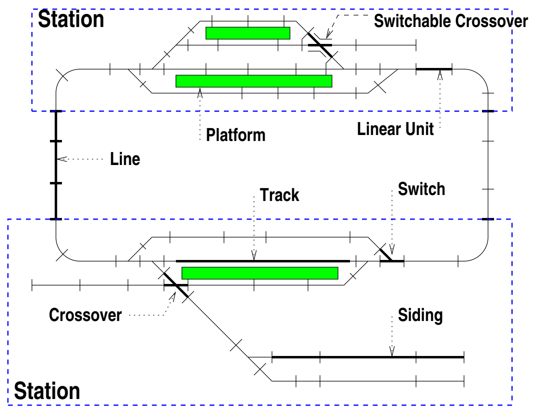

Fig. 1. A "Toy" Railway Net

2.1 The Basis

Rail nets consists of (rail) units. Units have connectors. Units of the net are connected "via" the connectors. Typically there are straight, ie., linear units (of two connectors), switches (points) (of typically three connectors), simple cross–overs (of four connectors), switchable cross–over (also of four connectors), etcetera. Some pairs of connectors form paths through the unit.

Figure 2 shows 'abstracted' examples of common rail units.

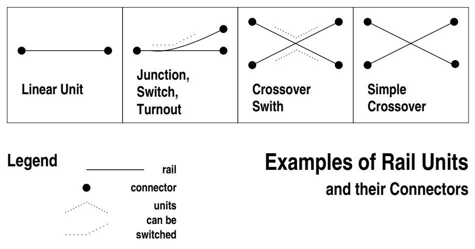

Fig. 2. Some Typical Rail Units

```
type
    N, U, C
    P = {| (c,c′):(C×C) • c≠c′ |}
value
    obs_Us: N → U-set
    obs_Cs: U → C-set
    obs_Ps: U → P-set
axiom
    ∀ n:N,u:U,c,c′:C,p:P •
        u ∈ obs_Us(n) ⇒
            (c,c′) ∈ obs_Ps(u) ⇒ {c,c′} ⊆ obs_Cs(u)
```

[5] In the following we focus just on units, for several sections, then on timed units, that is: Functions from time to units. In earlier papers — related to railway modelling in the style and for purposes similar to the ones of the current paper — we have given models of other facets of railways: Bjørner et al. (1999a) deals with train scheduling, Bjørner (2000) gives a comprehensive model of railway nets — and is a good prerequisite for a deeper study of the present paper[6], while Bjørner et al. (2002) outlines metodological principles and some techniques for modelling railway nets, etc. Conference (CD ROM) publications Bjørner et al. (1999b,c) present larger scope models of railway systems.

2.2 Stable, Transition and Re–organisation States

A unit is at any one time either in a stable state, or in a transition state, or in a reconfiguration state. A rail unit event is one where a rail unit changes from one kind of state to another: From a stable state to a transition or a reconfiguration state; or from a transition or a reconfiguration state to a stable state. In all: Three kinds of states and four kinds of events.

We have decided to model "transitions" from stable states to stable states as not taking place instantaneously, but having some time duration. During that time of change we say that the rail unit is in a transition state.

Reconfiguration states are like transition states, but, in addition, the rail units changes basic characteristics.

2.2.1. Time and State Durations Units remain in stable, transition and reconfiguration states "for some time" ! We decide to endow each unit with possibly different minimum stable state, and maximum transition and reconfiguration state durations: A unit, irrespective of its state, must remain in any stable state for a minimum duration of time. A unit, irrespective of its state, at most remains in any transition state for a maximum duration of time. A unit, irrespective of its state, at most remains in any reconfiguration state for a maximum duration of time. The stable state minimum duration is (very much) larger than the maximum re–configuration duration, which again is (very much) larger than the maximum transition duration.

```
type
    T /∗ T is some limited, dense time range ∗/
    Δ /∗ δ: Δ is some time duration ∗/
```

[5] Annotation: P stand for the type, ie., the value class of all pairs of different connectors. N, U, and C stand for the types of nets, units and connectors, respectively. These latter types (N, U, C) are defined as sorts, hence we postulate some observers: Of units of a net, of connectors of a unit, and of paths of a unit. These observers are subject to certain constraints, ie., are defined through some axioms: For all nets it must be the case that for all units of such nets, it must be the case that for all paths of such units (of such nets) the two connectors of the path are connectors of the unit.

[6] The essence of the model of Bjørner (2000) is included here as appendix A.

```
    $s\Delta = \Delta$, $t\Delta = \Delta$, $r\Delta = \Delta$
**value**
    $lo\_T: U \rightarrow T$
    $obs\_\ell\_T: U \rightarrow T$
    $obs\_s\Delta: U \rightarrow s\Delta$,
    $obs\_t\Delta: U \rightarrow t\Delta$,
    $obs\_r\Delta: U \rightarrow r\Delta$
    $\ll,<,>,\gg: \Delta \times \Delta \rightarrow \textbf{Bool}$
    $\ll,<,>,\gg: T \times T \rightarrow \textbf{Bool}$
    $+,-: T \times \Delta \rightarrow T$
    $*: \Delta \times \textbf{Real} \rightarrow \Delta$  **pre** $\delta*r: r>0$
**axiom**
    $\forall$ u:U $\bullet$
        $obs\_s\Delta(u) \gg obs\_r\Delta(u) \gg obs\_t\Delta(u)$,
    $\forall\ 1\delta,2\delta:\Delta$ $\bullet$
        $1\delta \ll 2\delta \Rightarrow 1\delta < 2\delta \wedge 1\delta \gg 2\delta \Rightarrow 1\delta > 2\delta$
```

[7]

2.2.2. Stable States A stable state (of a unit) is a possibly empty set of pairs of connectors of that unit. At any one time, when in a stable state, a unit is willing to be in any one of a number of states, its (current) state space. If a pair of connectors is in some stable state then that means that a train can move across the unit in the direction implied by the pairing: from the first connector to the second connector. A unit in a stable state has been so for a duration — which we assume can be observed.

Figure 3 shows two kinds of rail units and the possible stable states they may 'occupy'.

The arrows of Figure 3 shall designate possible ("open") directions of (allowed, "free") movement. To be able to compare units, and to say that a unit at one time, in some state, "is the same" as a unit, at another time, in another state, we introduce a "normalisation" function: nor_Σ. It behaves as if it "resets" the current state of a unit to the empty state, and as if the elapsed time is "zero" — leaving all else unchanged.[8]

[7] Annotation: We can observe, lo_T, the time when a rail unit was "first" installed, ie., "started". We can observe, obs_ℓ_T, the time when a rail unit most recently entered a stable state. (Initial value of this observer function when applied to a rail unit, u, that has never left the stable state in which it was first "started" is "its start" time (lo_T(u)).) We can observe respective durations (minimum stable obs_sΔ, maximum transition obs_tΔ, and maximum reconfigurtion obs_rΔ durations) from a(ny) unit, and in any state. All are total functions. And we can speak of ordering relations smaller and larger, respectively very much smaller and very much larger ordering relations — such that if a duration is very much smaller [larger] then it is at least strictly smaller [strictly larger], etc. And we can perform "arithmtic"–like addition, subtraction and multiplication operations on appropriate time concepts: Time and durations. Details are given in the formula. These operations otherwise behave as you would expect them to.

[8] The latter is, however, not formalised. But ought be.

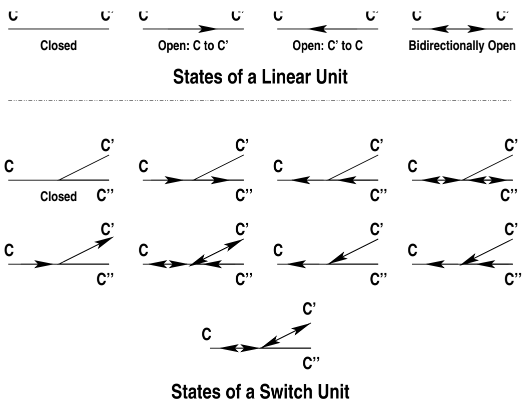

Fig. 3. Possible Stable States of Linear and Switch Units

```
**type**
    $PS = P\text{-}\textbf{set}$
    $s\Sigma = PS$
**value**
    $is\_s\Sigma: U \rightarrow \textbf{Bool}$
    $obs\_s\Sigma: U \overset{\sim}{\rightarrow} s\Sigma$
    $obs\_s\Omega: U \rightarrow s\Omega$
    $obs\_\Delta: U \rightarrow \Delta$
    $obs\_s\Delta: U \rightarrow s\Delta$
    $nor\_\Sigma: U \rightarrow U$
**axiom**
    $\forall$ u:U $\bullet$
        $is\_s\Sigma(u) \Rightarrow$
            $obs\_s\Sigma(u) \in obs\_s\Omega(u) \wedge$
            $obs\_s\Sigma(u) \subseteq obs\_Ps(u) \wedge$
            $obs\_s\Sigma(nor\_\Sigma(u)) = \{\}$
```

[9] Any one particular unit, "as it is 'laid down' in the landscape", may not be intended to 'occupy' all the states made possible by its connectors. Thus a linear unit in a railway marshalling yard, especially those "sloping" away, "down" from the hump, towards the yard itself, are usually provided with "braking gear" that would be wrecked if a freight car is forced "up the slop". We refer to Figure 4 on the following page — where a downwards slope is to the right from the hump.

2.2.3. Transition States When a unit is in a transition state it is making a transition from one stable state to another.

[9] Annotation: A stable state is a set of paths. A stable state space is a set of stable states. From a unit one can observe, it is postulated, whether it is in a stable state or not. From a unit in a stable state one can observe its (current) state, and one can always — in any state — observe its (current) state space. A hypothetical function, nor_Σ "projects" the current state and its current duration in that state onto the "void" state of no (open) paths and the "zero" duration. Not that the function obs_Δ is total, and is not to be confused with the likewise total function obs_sΔ. The latter yields the minimum duration that a unit must be in any stable state. The former yields the duration during which the unit has been in its current state — whether stable, transition or reconfiguration.

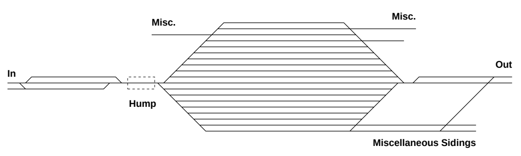

Fig. 4. A Marshalling Yard

We now make the following crucial modelling decision: Since we are dealing, throughout, with man–made phenomena, with entities most of whose properties we "design into" these physical "gadgets" we can assume the following: That we can observe from the rail units "their intention": Namely, in this case, that they are to make a transition from one, known, stable state to another, known, stable state, and that, at any one time of observing such a transition, we can also observe the elapsed time duration since the start of a transition.

type
  $\mathrm{t\Sigma}=\{|(\mathrm{s}'\sigma,\mathrm{s}''\sigma):(\mathrm{s\Sigma}\times\mathrm{s\Sigma})\bullet\mathrm{s}'\sigma\neq\mathrm{s}''\sigma|\}$
  $\mathrm{t\Omega} = \mathrm{t\Sigma}$**-set**
value
  $\mathrm{is\_t\Sigma}$: U → **Bool**
  $\mathrm{obs\_t\Sigma}$: U → $\mathrm{t\Sigma}$
  $\mathrm{obs\_t\Omega}$: U → $\mathrm{t\Omega}$
  $\mathrm{obs\_\Delta}$: U → Δ
  $\mathrm{obs\_t\Delta}$: U → $\mathrm{t\Delta}$
axiom
  $\forall\ \mathrm{u:U,s'\sigma,s''\sigma:\Sigma}\ \bullet$
   $\mathrm{is\_t\Sigma(u)} \Rightarrow (\mathrm{s}'\sigma,\mathrm{s}''\sigma) = \mathrm{obs\_t\Sigma(u)} \Rightarrow$
    $(\mathrm{s}'\sigma,\mathrm{s}''\sigma) \in \mathrm{obs\_t\Omega(u)}\wedge\{\mathrm{s}'\sigma,\mathrm{s}''\sigma\}\subseteq\mathrm{obs\_s\Omega(u)}$

[10] The dynamics of this change will be elaborated upon later. Suffice it to hint that the change from a stable state to the "beginning" of a transition state is an event, likewise is the change from a transition state to the stable state, and the stable state of the unit "just" before the transition state must be the same as the first stable state of the pair of the transition state, while the stable state of the unit "just" after the transition state must be the same as the second stable state of the pair of the transition state. [11]

2.2.4. Reconfiguration States A rail unit may be subject to reconfiguration: In a net some existing (ie., "old") rail units need be "changed" by allowing "additional", or dis–allowing "previously valid" paths, hence changing the state space, or by allowing new kinds of transitions, or both. Reconfiguration additionally permits new units to be "connected" to existing units' "dangling" connectors.

A rail unit reconfiguration thus changes its state space — from a past to a future state space, and therfore also by changing into a future transition state space, while possibly changing the unit from one stable state (of the past state space) to another (of the future state space) — where we impose the seemingly arbitrary constraint that the transition state (ie., the pair of before and after stable states) must be in both the "old" and the "new" set of transition states.

type
  $\mathrm{r\Sigma}' = (\mathrm{s\Omega}\times\mathrm{s\Sigma}\times\mathrm{t\Omega})\times(\mathrm{t\Omega}\times\mathrm{s\Sigma}\times\mathrm{s\Omega})$
  $\mathrm{r\Sigma} = \{|\ \mathrm{r}\sigma:\mathrm{r\Sigma}'\ \bullet\ \mathrm{wf\_r\Sigma(r\sigma)}\ |\}$
value
  $\mathrm{wf\_r\Sigma}$: $\mathrm{r\Sigma}'$ → **Bool**
  $\mathrm{wf\_r\Sigma}((\mathrm{s}'\omega,\mathrm{s}'\sigma,\mathrm{t}'\omega),(\mathrm{t}''\omega,\mathrm{s}''\sigma,\mathrm{s}''\omega)) \equiv$
    $\mathrm{s}'\sigma \in \mathrm{s}'\omega \wedge \mathrm{s}''\sigma \in \mathrm{s}''\omega \wedge$
    $(\mathrm{s}'\sigma,\mathrm{s}''\sigma) \in \mathrm{t}'\omega \cap \mathrm{t}''\omega \wedge$
    $\bigcup\mathrm{s}'\omega \cup \bigcup\mathrm{s}''\omega\subseteq\mathrm{obs\_Ps(u)} \wedge$
    $\forall\ (\mathrm{sa}\sigma,\mathrm{sb}\sigma):\mathrm{t\Sigma}\ \bullet$
     $(\mathrm{sa}\sigma,\mathrm{sb}\sigma) \in \mathrm{t}'\omega \Rightarrow \{\mathrm{sa}\sigma,\mathrm{sb}\sigma\} \subseteq \mathrm{s}'\omega \wedge$
     $(\mathrm{sa}\sigma,\mathrm{sb}\sigma) \in \mathrm{t}''\omega \Rightarrow \{\mathrm{sa}\sigma,\mathrm{sb}\sigma\} \subseteq \mathrm{s}''\omega$
  $\mathrm{is\_r\Sigma}$: U → **Bool**
  $\mathrm{obs\_r\Sigma}$: U $\stackrel{\sim}{\to}$ $\mathrm{r\Sigma}$
  $\mathrm{obs\_\Delta}$: U → Δ
  $\mathrm{obs\_r\Delta}$: U → $\mathrm{r\Delta}$

[12] We thus see that a reconfiguration state embodies also a transition state. And thus we inherit many of the constraints expressed earlier. Now they are part of the well–formedness of any reconfiguration state. For the other state types sorts were constrained via the axioms. A number of decisions have been made: We have decided, in this model, to maintain "redundant" "information": The before and after stable state spaces, as well as transition state spaces. And we have decided to impose a further "commonality" constraint: The actual state transition taken ("undergone") dur-

[10] Annotation: A transition state is a pair of different stable states. A transition state space is a set of (allowable) transition states. One can observe whether a unit is in a transition state or not. From a unit in a transition state one can observe its transition state. For all units which are in a transition state it is the case that the transition state is an allowable transition, and that the before and after stable states are in the (current) state space of the unit.

[11] We allow this seeming redundancy of representation in order to simplify some subsequent formalisations.

[12] Annotation: We observe that a reconfiguration state consists of a pair of "reversed" triples: The before and after reconfiguration stable state spaces (listed "outermost"), the before and after transition state spaces (listed "middle–most"), and the before and after stable states (listed "innermost" — all wrt. the "comma" that separates the two triples). For a reconfiguration state to be well–formed the before and after stable states must be in respective ("old", resp. "new") stable state spaces; the transition state formed by these must be in both the before and in the after transition state spaces; the set of paths of both the stable state spaces before and after must be in the paths of the unit — which thus does not change (!); and for all before [after] stable states of the transitions must be in the before [resp. after] stable state spaces. We may have to remove constraint that the total path set does not change under reconfiguration: Presently it is a refutable assertion.

ing reconfiguration must be one that was allowed before, as well as being allowed after, reconfiguration.

2.3 Dynamical Units

A railway net of many units, all timed to the same clock and time period, can be considered ideally an programmed, dynamic active system, less ideally, a dynamic reactive system.

These terms 'programmed, dynamic active system', respectively 'dynamic reactive system' are, for the realm of computing science and software engineering, that is: Programming methodology, described in Jackson (1995).

In this section we shall consider railway nets to be 'programmed'. That is: It is us, the managers of railway nets, who control the time–wise behaviour of the net — to a first approximation. To a second approximation, when ordering the rail units to undergo a reconfiguration and/or a transition, such changes may involve a time duration, such as modelled above. During those durations the rail units behave reactively: Over the time period of the duration they "switch state" in reaction to a control signal.

Although we shall thus primarily consider railway nets as programmed, active dynamic systems, we shall bring a model which appears to model railway nets as more general dynamic, active systems. But one should understand these models appropriately: As reflecting what can be observed from outside the system of railway nets plus their control. We shall subsequently review the above distinctions.

The behaviour of a unit, as seen from outside the railway net and its control, is that it changes from being in stable states and making transitions between these. A state transition is from the stable state before to the stable state after the transition. The stable state components of transition states must be in the current state space. A reconfiguration state transition has its stable states be in the intersection of, ie., in both, the before and after stable state spaces. (This constraint has already been formally expressed.)

2.3.1. Timed Units We now "lift" a unit to be a timed unit: That is, a function from time, in some dense interval, to "almost the same" unit ! We assume that we can delimit time intervals so that each timed unit is described as from some lower (lo_T) time upwards !

```
type
  T /* T is some downward bounded, */
     /* dense time range */
  TU = T → U
value
  lo_T: TU → T
  ℓ_T: (TU|U) → T
axiom
  ∀ tu:TU • unique_TU(tu)
value
  unique_TU: TU → Bool
  unique_TU(tu) ≡
   ∀ t,t′:T •
      {t,t′}⊆𝒟 tu ∧
      no_rΣs(tu)(t,t′)⇒same_Us({tu(t),tu(t′)})

  no_rΣs: TU → (T × T) → Bool
  no_rΣs(tu)(t,t′) ≡
   is_sΣ(tu(t)) ∧ is_sΣ(tu(t′)) ∧
   ∼∃ t″:T • t<t″<t′ ∧ is_rΣ(tu(t″))

  same_Us: U-set → Bool
  same_Us(us) ≡
   ∀ u,u′:U •
      obs_sΣ(nor_U(u)) = obs_sΣ(nor_U(u′)) ∧
      us ⊆ obs_sΩ(u)
   assert: ∀ u″:U•u″ ∈ us ⇒ us ⊆ obs_sΩ(u″)
```

[13] tu:TUs are continuous functions over their lower limited, although infinite definition set of times.

2.3.2. Operations on Timed Units In the following we will abstract from the two operations that are implied by the transition state, and the reconfiguration state. That is: We think, now, of these states as having been brought about by controls, ie., by external events and communication between an environment and the net (or, as in the case of timed rail units, between an environment and respective units).

So an operation on a timed unit is something that takes place, as some time, say τ, and which involves an operator. The meaning of the operator is what we model, not the syntax that is eventually needed in order to concretely implement the operation. And that meaning we take to involve the following entities: A function, ϕ, which is like a timed unit, except that its lower time limit is like "0". And a time duration, oδ, for the operation.

The idea is now that applying an operation ϕ at time τ, means that the timed unit function, tu, is "extended" by " glueing" the operation function

[13] Annotation: The lo_T and ℓ_T observers yield the lowest time recorded for the timed unit, respectively the ℓast time a state change took effect. With the current time we can always know how much time has elapsed since such a state change. For any timed unit it is the case it reflects a unique, a "same" unit. That is: that for all times in the definition set of the timed unit such that there are no "intervening" reconfiguration states, it is the case that the units are "the same".

ϕ to tu "chopped" at τ. After the operation has completed, at time τ+oδ, the unit remains in the state it was left in by ϕ at the end of its completion.

```
value
    lo_δ:Δ, hi_δ:Δ
axiom
    [ lo_δ ‵‵behaves like zero″ ]
type
    Θ
    Φ = ΦΔ → U
        /∗ ΦΔ is a continuous ∗/
        /∗ relative time interval ∗/
    ΦΔ = {loδ..hiδ}
        /∗ The above is not proper RSL ∗/
value
    obs_Φ: Θ → Φ
    obs_oΔ: Θ → {hi_δ .. lo_δ}
    OP: Θ → TU → T → TU
    OP(θ)(tu)(τ) ≡
        let φ = obs_Φ(θ),
            oδ = obs_oΔ(θ) assert: oδ=hi_δ−lo_δ,
            lo_t = lo_T(tu) in
        λ t:T • if t<lo_t then chaos
            elsif lo_t≤t≤τ then tu(t)
            elsif τ<t<τ+oδ then φ(t−τ)
            elsif t≥τ+oδ then φ(oδ) end end
```

[14] In the above — generalised — formulation of the effect of operations on timed units we have abstracted from whether these "stood" for state transitions or state reconfigurations. We have alkso made a number of general assumptions. These we now describe and formalise: The initial unit of the operation must be compatible with (for simplicity we here take it to be: the same as) the unit of the timed unit at the time the operations is applied.

```
    OP(θ)(tu)(τ) ≡ ...
    pre obs_Φ(θ)(lo_δ) = tu(τ)
```

One can think of the following constraint being already "syntactically" expressed in the specification of transition and reconfiguration states. We refer to Section 2.2.3 and Section 2.2.4. These state change specifications ("redundantly") specified the "before" and "after" states, where specifying the "after", ie., the final state, would have sufficed.

We leave it to another occassion to provide a proper linkage between specifying the syntactics of the operations and the already specified state change types.

2.3.3. State Sequences In the previous section we view timed units as something that changed only as the result of applying operations to the (timed) units. In this section we shall revert to looking at timed units as entities that has observable behaviour — ie., which can be observed from a vantage point "outside" the units and the "control machinery" that effects the operations.

Any one unit resides in a sequence of "adjacent" states: (i) For some time in a stable state, ψ, (ii) then, perhaps for a short time in a transation state: $\psi \mapsto \psi'$, (iii) then, as (i–ii): for some time in ψ', then $\psi' \mapsto \psi''$, etc.: (iv) ψ'', $\psi'' \mapsto \psi'''$, ψ''', $\psi''' \mapsto \psi''''$, etcetera. Maybe after a very long time compared to the time span from stable state ψ to stable state $\psi''''\cdots'$, the unit goes into a reconfiguration state. Whereupon (i–iv) is repeated, for a possibly other stable state and transition state sequence. One constraint that rules state changes with respect to state transitions (and, of course, stable states) is expressed below:

```
axiom
    ∀ tu:TU •
        ∀ t:T • t ∈ 𝒟 tu ⇒
            is_tΣ(tu(t)) ⇒
                is_sΣ(tu(t−obs_tΔ(tu(t)))) ∧
                is_sΣ(tu(t+obs_tΔ(tu(t)))) ∧
                let (s′σ,s″σ) = obs_tΣ(tu(t)) in
                s′σ = obs_sΣ(t−obs_tΔ(tu(t))) ∧
                s″σ = obs_sΣ(t+obs_tΔ(tu(t))) ∧
                {s′σ,s″σ} ⊆ obs_Ω(tu(t))
                /∗ last property follows ∗/
                /∗ from earlier axiom ∗/
                end
```

[15] We can formalise a similar constraint for the dynamic behaviour of units before and after undergoing, ie., residing in, reconfiguration states. We will leave that as an exercise.

[14] Annotation: We express the new timed unit function as a function of the old, namely tu, and the operation function, ϕ. The "new", "updated" timed unit behaves as follows: It is not defined for times earlier than the lowest time for which tu was defined. It behaves a tu for times between that lowest time and the present time, τ. For the next time interval, namely for the duration of the operation, it behaves as ϕ, and thereafter it remains stable ("until a next operation" is applied"). The above is expressed in the λ–Notation: λt:T•$\mathcal{E}$(args) dentes the function which when applied to arguments vals behaves as is expressed by the expression $\mathcal{E}$(args) where vals have been substituted for args.

[15] Annotation: If, at some time, t, a unit is in a transition state, then it is in a stable state both before and after that time by an amount which can be derived from, ie., is the transition state duration. And the stable states of the time unit at those times, before and after, are as prescribed by observing the transition state of the unit at that "some" time (t). Finally, as formalised before, the two stable states given as poart of the transition state are indeed in the unit's current staable state space.

2.4 Dynamical Nets

Railway nets consists of units — and otherwise possess many other properties. We now "lift" the conglomeration of all timed units to one timed net. This has to be understood as follows: Not only does the thus timed net consist of timed units but also of other "things".

2.4.1. Timed Nets Railway nets consists of units (and possibly more). A timed net is now a continuous function from time to nets. From a timed net (as from units and timed units) we can observe "its" lowest (its "begin" or "start") time.

```
type
   N, U, T
   TU = T → U
   TN = T → N
value
   lo_T: (U|TU|TN) → T
   obs_Us: N → U-set
```

For the purposes of our ensuing discussion we make the following simplifying, but not substantially limiting assumptions: For a given timed net, at any time after its "begin" time, it contains the same units as when first "started".

```
   assume: TN → T → Bool
   assume(tn)(τ) ≡
      ∀ t:T•lo_T(tn)<t≤τ ⇒
         nor_Us(tn(lo_T(tn)))=nor_Us(tn(t))

   nor_Us: N → U-set
   nor_Us(n) ≡
      { nor_U(u) | u:U • u ∈ obs_Us(n) }

   nor_Us: TN → U-set
   nor_Us(tn) ≡
      ⋃ {nor_Us(tn(t))|t:T•lo_T(tn)≤t≤τ}
```

nor_Us defines an equivalence class over any set of "different" units.

2.4.2. Relations between Timed Nets and Timed Units From a timed net we can "construct" a set of timed units reflecting the timed behaviour of all the units of the timed net.

```
value
   TN_2_TUs: TN → TU-set
   TN_2_TUs(tn) ≡
      { λ t:T • if t<lo_T(tu) then chaos
              else capture_U(tn)(u)(t) end
         | u:U • u ∈ obs_Us(tn(lo_T(u))) }
      pre ∀ t:T • t>lo_T(tn) assume(tn)(t)

   capture_U: TN → U → T → U
   capture_U(tn)(u)(t) ≡
      let n′ = tn(t) in
      let us′ = obs_Us(n′) in
      let u′:U • u′ ∈ us′ ∧ nor_U(u′)=nor_U(u)
      in u′  end end end
```

[16] We can not, alas, define the inverse function:

```
value
   TUs_2_TN: TU-set → TN
   conjecture:
      ∀ tn:TN • ∀ t:T • t>lo_T(tn) assume(tn)(t)
         ⇒ TUs_2_TN(TN_2_TUs(tn)) = tn
```

The reason is that the net is more than the sum of all its units. Had we defined a net to just be the set of all units, then a TUs_2_TN could be defined which satisfies the conjecture. Why is a net more than the sum total of all its units ? The answer to that question can, for example, be found in Bjørner (2000)[17]: We also wish to be able to observe, from a net, The delineations between lines and stations, the embedding, within stations, of tracks within the units of the stations, *&c.*

2.4.3. Selecting Timed Units Given a timed net and a "prototype" rail unit, that is, a normalised rail unit, we sometimes have a need to find that unit in the net, or, rather, to find "its" timed version:

```
value
   select_TU: TN → U ~→ TU
   select_TU(tn)(u) ≡
      let tus = TN_2_TUs(tn) in
      if ∃ tu:TU •
         tu ∈ tus ∧
         nor_U(tu(lo_T(tu)))=nor_U(u)
         then
            let tu:TU •
               tu ∈ tus ∧
               nor_U(tu(lo_T(tu)))=nor_U(u) in
            tu end
         else chaos
      end end
```

2.4.4. Operations on Timed Nets We have, in Section 2.3.2, defined the general idea of operations on timed units. We now wish to examine what the meaning of these operations are in the context of timed nets. Suppose we could say: Performing an operation on a timed unit of a timed net only affects that timed unit, and not any of the other timed units of the timed net, then performing "that same" operation, somehow

[16] Annotation: Each timed unit is that function of time which for times larger than or equal to the net "start" time, captures the "same" unit in the net. The function TN_2_TUs constructs from all units of the net their timed unit.
[17] See appendix A.

identifying the unit, would have to express the above, as is done below:

```
type
  $\Theta$
value
  OP: $\Theta \rightarrow (\mathrm{TN} \times \mathrm{U}) \rightarrow \mathrm{T} \rightarrow \mathrm{TN}$
  OP: $\Theta \rightarrow \mathrm{TU} \rightarrow \mathrm{T} \rightarrow \mathrm{TU}$

  OP($\theta$)(tn,u)($\tau$) as tn$'$
    pre
      $\exists$ u$'$:U $\bullet$
        u$'$ $\in$ obs_Us(tn($\tau$))
        $\wedge$ nor_U(u$'$)=nor_U(u)
    post
      let tu:U = select_TU(tn)(u) in
          tus = TN_2_TUs(tn),
          tus$'$ = TN_2_TUs(tn$'$) in
      tus\tus$'$ = tus$'$\tus = {OP($\theta$)(tu)($\tau$)}
      $\wedge$ ... end
```

[18] The $\wedge$ ... part of the above pre/post characterisation of operations on timed units of a timed net refers to the fact that *the whole is more than the sum of its parts*, that is: There may be aspects of the net which are affected by an operation, but not captured ny the individual rail units.

2.5 Discussion of the Model:

A model of certain aspects of a railway net has been presented. We could have chosen many different ways of formulating this model.

Next we shall discuss two aspects: Why we have not spoken about the unique identification of units. And whether the model of time (and timing) is the right model.

2.5.1. Why no Unique Unit Identification ? Perhaps most controversially is our tacit decision not to endow rail units with a unique identification. It is indeed true that each rail unit is unique. It is unique simply by the choice of its connectors. We never made that explicit. But it is indeed contained in the model of railway nets we referred to earlier. See Bjørner (2000) and also Appendix A. We could have instead endowed each unit with a unique identifier, but then we would have to express a lot of "book–keeping" constraints to secure that the already existing uniqueness of rail units was not being interfered with by the additional "unique" identifier.

[18] Annotation: The two function signatures are "almost the same". The meaning of performing the operation on a timed net is expressed by a pre/post pair of predicates relating the before and after states of the timed net(s). The function is partial since there must exist a unit of some timed unit, say at the "start" whose normalised form is that of the argument (ie., the "identifying") unit. Since we can assume this, and since the assumption of no new timed units properly within the time span of the timed net, and no "old" timed units "disappearing along the way" — since that assumption still holds — we can find the identified timed unit. Except for that timed unit all the other timed units of the before and after nets are unchanged. The only change is the (operation) "updated" timed unit.

2.5.2. Is it the Right Model of Timing ? When time is involved in a phenomenon, a good advice, in computing science circles, is usually not to introduce time explicitly in the model till the latest possible step of development — if at all ! It is obviously not an advice we have followed. So: Why not ? For two reasons: The first is, that we would otherwise have modelled timing by means of some combination (Yong and George (1999); Haxthausen and Yong (2000)) of RSL, as we have used it, George et al. (1992, 1995), and explicit timing constructs of an extended RSL, Yong and George (1999), or, Haxthausen and Yong (2000), of any one, or more, of the Duration Calculi Chaochen et al. (1992, 1993); Chaochen and Xiaoshan (1993); Chaochen (1993); Chaochen and Hansen (2002 (2003). Either approach might have "complicated" the presentation of the notations — which we have kept as Annotations in footnotes. So — in anticipation of such a possible complication — we have "cowardly" refrained. The other reason for not choosing to also use the above mentioned blends of RSL and either explicit RSL extending timing constructs, or one or more of the Duration Calculi, is that we wish, in a separate publication to perform those experiments: Of using exactly such "extensions", and then compare the two–three approaches.

In other words: It may not be the right model that we have presented in the current paper. "Time (!) will tell !" (Pun intended.)

3. POSSIBLE RELATIONS TO CONTROL THEORY

The whole purpose of Section 2 has been to present a model of a domain that is of interest to both software engineering and control engineering. We have presented "one side of the coin", the computing science facets of the models of such domains. It now remains to put forward, informally, some ideas that might relate to control theory, and to suggest that classical ideas of control theory, or just plain, simple calculus (ie., the differential and integral calculi) — that ideas from these disciplines — might be of use in further extending the computational models that are encountered when developing software.

The crucial phenomenon that forces us, so to speak, to raise the issue of possible relations —

as far as the domain of transport goes — between computing science and control engineering is that of our model of traffic:

```
type
   Train, Pos
   TF = T → (N × (Train $\overrightarrow{m}$ Pos))
```

where T is time, N is the time–varying net, Train stands for trains, and Pos is the position of trains. The timed net follows from traffic:

```
value
   timed_net: TF → TN
   timed_net(tf) ≡
      λt:T•let (n,tps)=tf(t) in n end
```

In control engineering we are used to monitor and control the net and the trains. Here they are brought together in one model. Something that can be done by means of the techniques of computing science, but something that does not seem to be so easy, as here, to express in usual control theoretic ways.

For a given train, say of identity tn:Tn, we may wish to observe its dynamics:

```
type
   Tn
   Train
   TRAIN = T → (N × Train × Pos)
value
   obs_Tn: Train → Tn
   monitor_Train: Tn → TF → TRAIN
   monitor_Train(tn)(tf) ≡
      λt:T•(let (n,tps) = tf(t) in
             let (trn,pos) = select(tn)(tps(t)) in
            (n,trn,pos) end end)

   select: Tn → (Train $\overrightarrow{m}$ Pos) → (Train × Pos)
   select(tn)(tps) ≡
      let trn:Train •
           trn ∈ dom tps∧obs_Tn(trn)=tn in
      (trn,tps(trn)) end
```

We shall end our odyssey here. We have brought our railway model right up to the physical quantities that have usually been the province of control theory and control engineering.

4. CONCLUSION

It is (thus) time to conclude.

4.1 Summary: What has been Achieved ?

We have shown how computing science can model dynamic systems that — we claim — cannot be modelled in control theory.

4.2 Speculations: Future Work

It now remains to exploit this possibility. We grant — ourselves actually having had a proper training in control theory — that from here on there arer indeed very many problems of net and train monitoring and control that we prefer to express in control theoretic terms, using the conventional notations of calculus.

Hence we must provide a proper "interface" between the continuous functions as expressed in this paper, ie., in RSL, and those of proper control theory. That has not been done in this paper — or elsewhere.

What we are suggesting is that of providing, for RSL formula involving functions like here over time, such notions as monotonicity, continuity, linearity, etc. In other words: It would most probably be useful to introduce such further notions as differentiablity and integrability.

And from there we go on — along lines as for example suggested in the fascinating paper by Willems: Willems (1991).

5. ACKNOWLEDGEMENTS

The author gratefully acknowledges many years of fruitful railway modelling collaboration with Søren Prehn and Chris George, before, during, and after my stay as UN Director of UNU/IIST, the UN University's International Institute for Software Technology, Macau SAR, China (1991–1997).

He likewise gratefully acknowledges the encouragement by Eckehard Schnieder, Technical University of Brunswick, Germany, to submit this and other papers (Bjørner (2000); Bjørner et al. (2002)) to similar conferences.

6. BIBLIOGRAPHICAL NOTES

REFERENCES

Dines Bjørner. Formal Software Techniques in Railway Systems. In Eckehard Schnieder, editor, *9th IFAC Symposium on Control in Transportation Systems*, pages 1–12, Technical University, Braunschweig, Germany, 13–15 June 2000. VDI/VDE-Gesellschaft Mess– und Automatisieringstechnik, VDI-Gesellschaft für Fahrzeug– und Verkehrstechnik.

Dines Bjørner, Chris W. George, and Søren Prehn. Computing Systems for Railways — A Rôle for Domain Engineering. Relations to Requirements Engineering and Software for Control Applications. In *Integrated Design and Process Technology. Editors: Bernd Kraemer and John C. Petterson*, page 26 pages, P.O.Box 1299, Grand View, Texas 76050-1299, USA, 24–28 June 2002. Society for Design and Process Science.

Dines Bjørner, C.W. George, and S. Prehn. *Scheduling and Rescheduling of Trains*, chapter 8, pages 157–184. *Industrial Strength Formal Methods in Practice,* Eds.: Michael G. Hinchey and Jonathan P. Bowen. FACIT, Springer–Verlag, London, England, 1999a.

Dines Bjørner, Søren Prehn, and Chris W. George. Formal Models of Railway Systems: Domains. Bldg. 344, DK–2800 Lyngby, Denmark, September 23 1999b. Presented at the FME Rail Workshop on Formal Methods in Railway Systems, FM'99 World Congress on Formal Methods, Toulouse, France. Avaliable on CD ROM.

Dines Bjørner, Søren Prehn, and Chris W. George. Formal Models of Railway Systems: Requirements. Presented at the FME Rail Workshop on Formal Methods in Railway Systems, FM'99 World Congress on Formal Methods, Toulouse, France. Avaliable on CD ROM.

Zhou Chaochen. Duration Calculi: An Overview. Published in: *Formal Methods in Programming and Their Applications,* Conference Proceedings, June 28 – July 2, 1993, Novosibirsk, Russia; (Eds.: D. Bjørner, M. Broy and I. Pottosin) LNCS 736, Springer-Verlag, 1993, pp 36–59.

Zhou Chaochen and Michael R. Hansen. *Duration Calculus: A formal approach to real–time systems.* Monographs in Theoretical Computer Science. Springer–Verlag, 2002 (2003).

Zhou Chaochen, C. A. R. Hoare, and A. P. Ravn. A Calculus of Durations. *Information Proc. Letters*, 40 (5), 1992.

Zhou Chaochen and Yu Huiqun. A duration Model for Railway scheduling. Technical Report 24b, UNU/IIST, P.O.Box 3058, Macau, May 1994.

Zhou Chaochen, Anders P. Ravn, and Michael R. Hansen. An Extended Duration Calculus for Real-time Systems. Published in: *Hybrid Systems*, LNCS 736, 1993.

Zhou Chaochen and Li Xiaoshan. A Mean Value Duration Calculus. Published as Chapter 25 in *A Classical Mind*, Festschrift for C.A.R. Hoare, Prentice-Hall International, 1994, pp 432–451.

Chris George, Peter Haff, Klaus Havelund, Anne Haxthausen, Robert Milne, Claus Bendix Nielsen, Søren Prehn, and Kim Ritter Wagner. *The RAISE Specification Language.* The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1992.

Chris George, Anne Haxthausen, Steven Hughes, Robert Milne, Søren Prehn, and Jan Storbank Pedersen. *The RAISE Method.* The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1995.

Anne Haxthausen and Xia Yong. Linking DC together with TRSL. In *Proceedings of 2nd International Conference on Integrated Formal Methods (IFM'2000), Schloss Dagstuhl, Germany, November 2000*, number 1945 in Lecture Notes in Computer Science, pages 25–44. Springer-Verlag, 2000.

Michael A. Jackson. *Software Requirements & Specifications: a lexicon of practice, principles and prejudices.* ACM Press. Addison-Wesley Publishing Company, Wokingham, nr. Reading, England; E-mail: ipc@awpub.add-wes.co.uk, 1995.

J.C. Willems. Paradigms and puzzles in the theory of dynamical systems. *IEEE Trans. on Automatic Control*, 36(3):259–294, March 1991.

Xia Yong and Chris W. George. An Operational Semantics for Timed RAISE. In Jeannette M. Wing, Jim Woodcock, and Jim Davies, editors, *FM'99 — Formal Methods*, pages 1008–1027. Springer–Verlag, volume 1709 of *LNCS: Lecture Notes in Computer Science*, 1999.

Appendix A. A RAILWAY TOPOLOGY MODEL

From Bjørner (2000).

A.1 The Structure

A.1.1. Narrative

We introduce the phenomena of railway nets, lines, stations, tracks, (rail) units, and connectors.

(1) A railway net consists of one or more lines and two or more stations.
(2) A railway net consists of rail units.
(3) A line is a linear sequence of one or more linear rail units.
(4) The rail units of a line must be rail units of the railway net of the line.
(5) A station is a set of one or more rail units.
(6) The rail units of a station must be rail units of the railway net of the station.
(7) No two distinct lines and/or stations of a railway net share rail units.
(8) A station consists of one or more tracks.
(9) A track is a linear sequence of one or more linear rail units.
(10) No two distinct tracks share rail units.
(11) The rail units of a track must be rail units of the station (of that track).
(12) A rail unit is either a linear, or is a switch, or a is simple crossover, or is a switchable crossover, etc., rail unit.
(13) A rail unit has one or more connectors.
(14) A linear rail unit has two distinct connectors, a switch rail unit has three distinct connectors, crossover rail units have four distinct connectors (whether simple or switchable), etc.
(15) For every connector there are at most two rail units which have that connector in common.
(16) Every line of a railway net is connected to exactly two, distinct stations of that railway net.
(17) A linear sequence of (linear) rail units is a non-cyclic sequence of linear units such that neighbouring units share connectors.

A.1.2. Formalisation

```
type
   N, L, S, Tr, U, C
value
   1.  obs_Ls: N → L-set,
   1.  obs_Ss: N → S-set
   2.  obs_Us: N → U-set,
   3.  obs_Us: L → U-set
   5.  obs_Us: S → U-set,
   8.  obs_Trs: S → Tr-set
   12.  is_Linear: U → Bool,
   12.  is_Switch: U → Bool
   12.  is_Simple_Crossover: U → Bool,
   12.  is_Switchable_Crossover: U → Bool
   13.  obs_Cs: U → C-set

   17.  lin_seq: U-set → Bool
       lin_seq(q) ≡
           let us = obs_Us(us) in
           ∀ i:U • u ∈ us ⇒ is_Linear(u) ∧
           ∃ q:U* • len q = card us ∧ elems q = us ∧
               ∀ i:Nat • {i,i+1} ⊆ inds q ⇒ ∃ c:C •
                   obs_Cs(q(i)) ∩ obs_Cs(q(i+1)) = {c} ∧
               len q > 1 ⇒
                   obs_Cs(q(i)) ∩ obs_Cs(q(len q)) = {}
               end
```

Some formal axioms are now given, not all !

```
axiom
1. ∀ n:N • card obs_Ls(n) ≥ 1,

1. ∀ n:N • card obs_Ss(n) ≥ 2,

3. ∀ n:N, l:L • l ∈ obs_Ls(n) ⇒ lin_seq(l)

4. ∀ n:N, l:L • l ∈ obs_Ls(n) ⇒ obs_Us(l) ⊆ obs_Us(n)

5. ∀ n:N, s:S • s ∈ obs_Ss(n) ⇒ card obs_Us(s) ≥ 1

6. ∀ n:N, s:S • s ∈ obs_Ls(n) ⇒ obs_Us(s) ⊆ obs_Us(n)
```

```
7.  ∀ n:N, l,l′:L •
        {l,l′} ⊆ obs_Ls(n) ∧ l≠l′
            ⇒ obs_Us(l) ∩ obs_Us(l′) = {}

7.  ∀ n:N, l:L, s:S •
        l ∈ obs_Ls(n) ∧ s ∈ obs_Ss(n)
            ⇒ obs_Us(l) ∩ obs_Us(s) = {}

7.  ∀ n:N, s,s′:S •
        {s,s′} ⊆ obs_Ss(n) ∧  s≠s′
            ⇒ obs_Us(s) ∩ obs_Us(s′) = {}

8.  ∀ s:S • card obs_Trs(s) ≥ 1

9.  ∀ n:N, s:S, t:T •
        s ∈ obs_Ss(n) ∧ t ∈ obs_Trs(s) ⇒ lin_seq(t)

10.  ∀ n:N, s:S, t,t′:T •
        s ∈ obs_Ss(n) ∧ {t,t′} ⊆ obs_Trs(s) ∧ t≠t′
            ⇒ obs_Us(t) ∩ obs_Us(t′) = {}

15. ∀ n:N • ∀ c:C •
        c ∈ ∪ { obs_Cs(u) | u:U • u ∈ obs_Us(n) }
            ⇒ card{ u | u:U •
                  u ∈ obs_Us(n) ∧ c ∈ obs_Cs(u) } ≤ 2

16. ∀ n:N,l:L •  l ∈ obs_Ls(n) ⇒
      ∃ s,s′:S • {s,s′} ⊆ obs_Ss(n) ∧ s≠s′ ⇒
         let sus = obs_Us(s),
             sus′ = obs_Us(s′),
             lus = obs_Us(l) in
         ∃ u:U • u ∈ sus, u′:U •
                u′ ∈ sus′, u″,u‴:U • {u″,u‴} ⊆ lus •
                 let scs = obs_Cs(u), scs′ = obs_Cs(u′),
                 lcs = obs_Cs(u″), lcs′ = obs_Cs(u‴) in
                 ∃ ! c,c′:C •
                      c≠c′ ∧ scs ∩ lcs={c} ∧ scs′ ∩ lcs′={c′}
            end end
```

A.2 The Dynamics

A.2.1. Narrative

We introduce defined concepts such as paths through rail units, state of rail units, rail unit state spaces, routes through a railway network, open and closed routes, trains on the railway net, and train movement on the railway net.

(18) A path, $p : P$, is a pair of connectors, (c, c'),
(19) which are distinct,
(20) and of some unit.[19]
(21) A state, $\sigma : \Sigma$, of a unit is the set of all open paths of that unit (at the time observed).[20]
(22) A unit may, over its operational life, attain any of a (possibly small) number of different states ω, Ω.
(23) A route is a sequence of pairs of units and paths —
(24) such that the path of a unit/path pair is a possible path of some state of the unit, and such that "neighbouring" connectors are identical.
(25) An open route is a route such that all its paths are open.
(26) A train is modelled as a route.
(27) Train movement is modelled as a discrete function (ie., a map) from time to routes
(28) such that for any two adjacent times the two corresponding routes differ by at most one of the following:
  (a) a unit path pair has been deleted (removed) from one end of the route;
  (b) a unit path pair has been deleted (removed) from the other end of the route;
  (c) a unit path pair has been added (joined) from one end of the route;
  (d) a unit path pair has been added (joined) from the other end of the route;
  (e) a unit path pair has been added (joined) from one end of the route, and another unit path par has been deleted (removed) from the other end of the route;
  (f) a unit path pair has been added (joined) from the other of the route, and another unit path par has been deleted (removed) from the one end of the route;
  (g) or there has been no changes with respect to the route (yet the train may have moved);
(29) and such that the new route is a well–formed route.

We shall arbitrarily think of "one end" as the "left end", and "the other end", hence, as the "right end" — where 'left', in a model where elements of a list is indexed from 1 to its length, means the index 1 position, and 'right' means the last index position of the list.

A.2.2. Formalisation

```
type
18.  P′ = C × C
19.  P = {| (c,c′):P′ • c≠c′ |}
21.  Σ = P-set
22.  Ω = Σ-set
23.  R′ = (U × P)*
24.  R ={| r:R′ • wf_R(r) |}
26.  Trn = R
27.  Mov′ = T →m Trn
28.  Mov = {| m:Mov′ • wf_Mov(m) |}
21.  obs_Σ: U → Σ
22.  obs_Ω: U → Ω

axiom
    ∀ u:U •
        let ω = obs_Ω(u),
              σ = obs_Σ(u) in
        σ ∈ ω ∧ 20.
        let cs = obs_Cs(u) in
        ∀ (c,c′):P • (c,c′) ∈ ∪ ω
            ⇒ {c,c′} ⊆ obs_Cs(u)
        end end

24.  wf_R: R′ → Bool
     wf_R(r) ≡
         len r > 0 ∧
         ∀ i:Nat • i ∈ inds r
             let (u,(c,c′)) = r(i) in
             (c,c′) ∈ ⋃ obs_Ω(u) ∧ i+1 ∈ inds r
                 ⇒ let (_,(c″,_)) = r(i+1) in c′ = c″
         end end

25.  open_R: R → Bool
     open_R(r) ≡
         ∀ (u,p):U×P • (u,p) ∈ elems r ∧ p ∈ obs_Σ(u)

27.  wf_Mov: Mov → Bool
     wf_Mov(m) ≡ card dom m ≥ 2 ∧
         ∀ t,t′:T • t,t′ ∈ dom m ∧ t < t′
         ∧ adjacent(t,t′) ⇒
             let (r,r′) = (m(t),m(t′))
                 (u,p):U×P • p ∈ ⋃ obs_Ω(u) in
     28a.  (l_d(r,r′,(u,p))    ∨   28b.  r_d(r,r′,(u,p))   ∨
     28c.  l_a(r,r′,(u,p))     ∨   28d.  r_a(r,r′,(u,p))   ∨
     28e.  l_d_r_a(r,r′,(u,p)) ∨   28f.  r_d_l_a(r,r′,(u,p))∨
     28g.  r=r′) ∧ wf_R(r′)
             end
```

The last line's route well–formedness ensures that the type of Move is maintained.

```
value
    adjacent: T × T → Bool
    adjacent(t,t′) ≡ ∼∃ t″:T • t″ ∈ dom m ∧ t < t″ < t′

    l_d,r_d,l_a,r_a,l_d_r_a,r_d_l_a: R × R × P → Bool

    l_d(r,r′,(u,p)) ≡ r′ = tl r      pre len r>1
    r_d(r,r′,(u,p)) ≡ r′ = fst(r)   pre len r>1
    l_a(r,r′,(u,p)) ≡ r′ = ⟨(u,p)⟩⌢r
    r_a(r,r′,(u,p)) ≡ r′ = r⌢⟨(u,p)⟩
    l_d_r_a(r,r′,(u,p)) ≡ r′ = tl r⌢⟨(u,p)⟩
    r_d_l_a(r,r′,(u,p)) ≡ r′ = ⟨(u,p)⟩⌢fst(r)

    fst: R ~→ R′
    fst(r) ≡ ⟨ r(i) | i in ⟨1..len r−1⟩ ⟩
```

If r as argument to fst is of length 1 then the result is not a well–formed route, but is in R′.

[19] A path of a unit designate that a train may move across the unit in the direction from c to c'. We say that the unit is open in the direction of the path.
[20] The state may be empty: the unit is closed.